HDNUM
Heidelberger Numerikbibliothek

Peter Bastian
Universität Heidelberg,
Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
Im Neuenheimer Feld 368, D-69120 Heidelberg
Peter.Bastian@iwr.uni-heidelberg.de

14. November 2016

Zusammenfassung

Die Heidelberger Numerikbibliothek wurde begleitend zu den Vorlesungen *Einführung in die Numerik* und *Numerik* in der Programmiersprache C++ entwickelt und stellt einfach zu benutzende Klassen für grundlegende Aufgaben in der Numerik bis hin zur Lösung von gewöhnlichen Differentialgleichungen zur Verfügung. In fast allen Klassen ist der benutzte Zahlentyp parametrisierbar so dass auch hochpräzise Rechnungen durchgeführt werden können.

1 Einführung

1.1 Was ist HDNUM

Die Heidelberger Numerikbibliothek (HDNUM) ist eine C++-basierte Bibliothek zur Durchführung der praktischen Übungen zu den Vorlesungen *Einführung in die Numerik* und *Numerik (gewöhnlicher Differentialgleichungen)*. Die aktuelle Version ist unter

`https://parcomp-git.iwr.uni-heidelberg.de/Teaching/hdnum`

verfügbar und wird mit dem Versionskontrollsystem `git` verwaltet. Spezifische Versionen können auf der jeweiligen Vorlesungswebseite veröffentlicht werden.

Ziele bei der Entwicklung von HDNUM waren i) die einfache Benutzbarkeit (inklusive einfacher Installation), ii) die Demonstration objektorientierter Programmierung in der Numerik sowie die Möglichkeit zur Durchführung von Berechnungen mit beliebiger Genauigkeit auf Basis der Gnu Multiple Precision[1] Bibliothek. Derzeit bietet HDNUM die folgende Funktionalität:

1) Klassen für Matrizen und Vektoren

2) Lösung linearer Gleichungssystem

3) Lösung nichtlinearer Gleichungssysteme

4) Lösung von Systemen gewöhnlicher Differentialgleichungen

5) Lösung der Poissongleichung mit Finiten Differenzen

[1] `https://gmplib.org`

1.2 Installation

HDNUM ist eine „header only“ Bibliothek und erfordert keine Installation ausser dem Herunterladen der Dateien. Die aktuelle Version kann man mittels

```
$ git clone https://parcomp-git.iwr.uni-heidelberg.de/Teaching/hdnum.git
```

herunterladen. Hierzu ist das Programm `git` erforderlich, welches für alle Betriebssysteme frei erhältlich ist. Alternativ wird üblicherweise auch ein komprimiertes `tar`-archive auf der Homepage der jeweiligen Vorlesung angeboten. Dies entpackt man mittels

```
$ tar zxvf hdnum-XX.tgz
```

In dem entpackten bzw. installiertem Verzeichnis findet man die folgenden Dateien und Unterverzeichnisse:

- `hdnum.hh`: Diese Header-Datei ist in ein C++-Programm einzubinden um HDNUM nutzen zu können.
- Das Verzeichnis `mystuff` ist für ihre Programme vorgesehen aber Sie können natürlich jedes andere Verzeichnis nutzen. Wichtig ist nur, dass der Compiler die Datei `hdnum.hh` findet. Im Verzeichnis `mystuff` ist schon ein Beispielprogramm um gleich loslegen zu können. Dieses Programm übersetzt man mit:

  ```
  $ cd mystuff
  $ g++ -I.. -o example example.cc
  ```

  Diese Befehle setzen voraus, dass auf ihrem System der GNU C++-Compiler installiert ist. Unter Windows oder für andere Compiler müssen Sie die Befehle entsprechend anpassen.
- Das Verzeichnis `examples` im HDNUM-Ordner enthält viele Beispiele geordnet nach Programmierkurs, Numerik 0 und Numerik 1.
- Das Verzeichnis `src` im HDNUM-Ordner enthält den Quellcode der HDNUM Bibliothek. Diese Dateien werden von `hdnum.hh` eingebunden.
- Das Verzeichnis `programmierkurs` im HDNUM-Ordner enthält die Folien zum Programmierkurs.
- Das Verzeichnis `tutorial` im HDNUM-Ordner enthält den Quellcode für dieses Dokument.

GNU Multiple Precision Bibliothek

HDNUM kann Berechnungen mit hoher Genauigkeit durchführen. Hierzu ist die GNU Multiple Precision Bibliothek (GMP) erforderlich, welche Sie für viele Systeme kostenlos erhalten können. Um GMP nutzen zu können müssen Sie in der Datei `hdnum.hh` die Zeile

```
#define HDNUM_HAS_GMP 1
```

auskommentieren. Zusätzlich sind eventuell Compileroptionen notwendig damit der Compiler die Headerdateien und Bibliotheken von GMP findet. Dies kann dann so aussehen:

```
$ g++ -I.. -I/opt/local/include -o example example.cc -L/opt/local/lib -lgmpxx -lgmp
```

2 Lineare Algebra

2.1 Vektoren

`hdnum::Vector<T>`

- `hdnum::Vector<T>` ist ein Klassen-Template.
- Es macht aus einem beliebigen (Zahl-)Datentypen `T` einen Vektor.
- Auch komplexe und hochgenaue Zahlen sind möglich.
- Vektoren verhalten sich so wie man es aus der Mathematik kennt:
  - Bestehen aus n Komponenten.
  - Diese sind von 0 bis $n-1$ (!) durchnummeriert.
  - Addition und Multiplikation mit Skalar.
  - Skalarprodukt und Euklidsche Norm
  - Matrix-Vektor-Multiplikation
- Die folgenden Beispiele findet man in `vektoren.cc`

Konstruktion und Zugriff

- Konstruktion mit und ohne Initialisierung

```
hdnum::Vector<float> x(10);         // Vektor mit 10 Elementen
hdnum::Vector<double> y(10,3.14);   // 10 Elemente initialisiert
hdnum::Vector<float> a;             // ein leerer Vektor
```

- Speziellere Vektoren

```
hdnum::Vector<std::complex<double> >
  cx(7,std::complex<double>(1.0,3.0));
mpf_set_default_prec(1024); // Setze Genauigkeit fuer mpf_class
hdnum::Vector<mpf_class> mx(7,mpf_class("4.44"));
```

- Zugriff auf Element

```
for (std::size_t i=0; i<x.size(); i=i+1)
  x[i] = i;                  // Zugriff auf Elemente
```

- Vektorobjekt wird am Ende des umgebenden Blockes gelöscht.

Kopie und Zuweisung

- Copy-Konstruktor: Erstellen eines Vektors als Kopie eines anderen

```
hdnum::Vector<float> z(x); // z ist Kopie von x
```

- Zuweisung, auch die Größe wird überschrieben!

```
b = z;               // b kopiert die Daten aus z
a = 5.4;             // Zuweisung an alle Elemente
hdnum::Vector<double> w;   // leerer Vektor
w.resize(x.size()); // make correct size
w = x;               // copy elements
```

- Ausschnitte von Vektoren

```
hdnum::Vector<float> w(x.sub(7,3));// w ist Kopie von x[7],...,x[9]
z = x.sub(3,4);              // z ist Kopie von x[3],...,x[6]
```

Rechnen und Ausgabe

- Vektorraumoperationen und Skalarprodukt

```
w += z;             // w = w+z
w -= z;             // w = w-z
w *= 1.23;          // skalare Multiplikation
w /= 1.23;          // skalare Division
w.update(1.23,z);   // w = w + a*z
float s;
s = w*z;            // Skalarprodukt
```

- Ausgabe auf die Konsole

```
std::cout << w << std::endl;// schoene Ausgabe
w.iwidth(2);                // Stellen in Indexausgabe
w.width(20);                // Anzahl Stellen gesamt
w.precision(16);            // Anzahl Nachkommastellen
std::cout << w << std::endl;// nun mit mehr Stellen
std::cout <<cx << std::endl;// geht auch fuer complex
std::cout <<mx << std::endl;// geht auch fuer mpf_class
```

Beispielausgabe

```
[   0]    1.204200e+01
[   1]    1.204200e+01
[   2]    1.204200e+01
[   3]    1.204200e+01

[ 0] 1.20420007705688 48e+01
[ 1] 1.20420007705688 48e+01
[ 2] 1.20420007705688 48e+01
[ 3] 1.20420007705688 48e+01
```

Hilfsfunktionen

```
float d = norm(w);          // Euklidsche Norm
d = w.two_norm();           // das selbe
zero(w);                    // das selbe wie w=0.0
fill(w,(float)1.0);         // das selbe wie w=1.0
fill(w,(float)0.0,(float)0.1); // w[0]=0, w[1]=0.1, w[2]=0.2, ...
unitvector(w,2);            // kartesischer Einheitsvektor
gnuplot("test.dat",w);      // gnuplot Ausgabe: i w[i]
gnuplot("test2.dat",w,z);   // gnuplot Ausgabe: w[i] z[i]
```

Funktionen

- Beispiel: Summe aller Komponenten

```
double sum (hdnum::Vector<double> x) {
  double s(0.0);
  for (std::size_t i=0; i<x.size(); i=i+1)
    s = s + x[i];
  return s;
}
```

- Verbesserte Version mit **Funktionentemplate** und by-const-reference Übergabge

```
template<class T>
T sum (const hdnum::Vector<T>& x) {
  T s(0.0);
  for (std::size_t i=0; i<x.size(); i=i+1)
    s = s + x[i];
  return s;
}
```

- By-value Übergabe ist be großen Objekten vorzuziehen

2.2 Matrizen

`hdnum::DenseMatrix<T>`

- `hdnum::DenseMatrix<T>` ist ein Klassen-Template.
- Es macht aus einem beliebigen (Zahl-)Datentypen `T` eine Matrix.
- Auch komplexe und hochgenaue Zahlen sind möglich.
- Matrizen verhalten sich so wie man es aus der Mathematik kennt:
  - Bestehen aus $m \times n$ Komponenten.
  - Diese sind von 0 bis $m-1$ bzw. $n-1$ (!) durchnummeriert.
  - $m \times n$-Matrizen bilden einen Vektorraum.
  - Matrix-Vektor und Matrizenmultiplikation.
- Die folgenden Beispiele findet man in `matrizen.cc`

Konstruktion und Zugriff

- Konstruktion mit und ohne Initialisierung

```
hdnum::DenseMatrix<float> B(10,10);      // 10x10 Matrix uninitialisiert
hdnum::DenseMatrix<float> C(10,10,0.0); // 10x10 Matrix initialisiert
```

- Zugriff auf Elemente

```
for (int i=0; i<B.rowsize(); ++i)
  for (int j=0; j<B.colsize(); ++j)
    B[i][j] = 0.0;           // jetzt ist B initialisiert
```

- Matrixobjekt wird am Ende des umgebenden Blockes gelöscht.

Kopie und Zuweisung

- Copy-Konstruktor: Erstellen einer Matrix als Kopie einer anderen

```
hdnum::DenseMatrix<float> D(B); // D Kopie von B
```

- Zuweisung, kopiert auch Größe mit

```
hdnum::DenseMatrix<float> A(B.rowsize(),B.colsize()); // korrekte Groesse
A = B;                      // kopieren
```

- Ausschnitte von Matrizen (Untermatrizen)

```
hdnum::DenseMatrix<float> F(A.sub(1,2,3,4));// 3x4 Mat ab (1,2)
```

Rechnen mit Matrizen

- Vektorraumoperationen (Vorsicht: Matrizen sollten passende Größe haben!)

```
A += B;            // A = A+B
A -= B;            // A = A-B
A *= 1.23;         // Multiplikation mit Skalar
A /= 1.23;         // Division durch Skalar
A.update(1.23,B); // A = A + s*B
```

- Matrix-Vektor und Matrizenmultiplikation

```
hdnum::Vector<float> x(10,1.0); // make two vectors
hdnum::Vector<float> y(10,2.0);
A.mv(y,x);                 // y = A*x
A.umv(y,x);                // y = y + A*x
A.umv(y,(float)-1.0,x);    // y = y + s*A*x
C.mm(A,B);                 // C = A*B
C.umm(A,B);                // C = C + A*B
A.sc(x,1);                 // mache x zur ersten Spalte von A
A.sr(x,1);                 // mache x zur ersten Zeile von A
float d=A.norm_infty();    // Zeilensummennorm
d=A.norm_1();              // Spaltensummennorm
```

Ausgabe und Hilfsfunktionen

- Ausgabe von Matrizen

```
std::cout << A.sub(0,0,3,3) << std::endl;// öschne Ausgabe
A.iwidth(2);                 // Stellen in Indexausgabe
A.width(10);                 // Anzahl Stellen gesamt
A.precision(4);              // Anzahl Nachkommastellen
std::cout << A << std::endl;// nun mit mehr Stellen
```

- einige Hilfsfunktionen

```
identity(A);
spd(A);
fill(x,(float)1,(float)1);
vandermonde(A,x);
```

Beispielausgabe

```
                0           1           2           3
  0   4.0000e+00 -1.0000e+00 -2.5000e-01 -1.1111e-01
  1  -1.0000e+00  4.0000e+00 -1.0000e+00 -2.5000e-01
  2  -2.5000e-01 -1.0000e+00  4.0000e+00 -1.0000e+00
  3  -1.1111e-01 -2.5000e-01 -1.0000e+00  4.0000e+00
```

Funktion mit Matrixargument

Beispiel einer Funktion, die eine Matrix A und einen Vektor b initialisiert.

```
template<class T>
void initialize (hdnum::DenseMatrix<T>& A, hdnum::Vector<T>& b)
{
  if (A.rowsize()!=A.colsize() || A.rowsize()==0)
    HDNUM_ERROR("need␣square␣and␣nonempty␣matrix");
  if (A.rowsize()!=b.size())
    HDNUM_ERROR("b␣must␣have␣same␣size␣as␣A");
  for (int i=0; i<A.rowsize(); ++i)
```

```
    {
      b[i] = 1.0;
      for (int j=0; j<A.colsize(); ++j)
        if (j<=i) A[i][j]=1.0; else A[i][j]=0.0;
    }
}
```

2.3 LR Zerlegung

2.4 QR Zerlegung

Anhang A Kleiner Programmierkurs

Anhang B Unix Kommandos